Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №2

із дисципліни «Технології розроблення програмного забезпечення»
Тема: «Основи проектування**»**

Виконав:
студент групи ІА-31:
Кунда А.П.

Перевірив:
Мягкий М.Ю.

Київ- 2025

Тема: Основи проектування.
Мета: Обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

Завдання:
• Ознайомитись з короткими теоретичними відомостями.
• Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.
• Спроєктувати діаграму класів предметної області.
• Вибрати 3 варіанти використання та написати за ними сценарії використання.
• На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізуватишаблон Repository для взаємодії з базою даних.
• Нарисувати діаграму класів для реалізованої частини системи.
• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних.

Хід роботи

Тема проекту: JSON Tool (ENG) (strategy, command, observer, template method, flyweight)

Display JSON schema with syntax highlight. Validate JSON schema and display errors. Create user friendly table\list box\other for read and update JSON schema properties metadata (description, example, data type, format, etc.). Auto save\restore when edit, maybe history. Can check JSON value by schema (Put schema and JSON = valid\invalid, display errors). Export schema as markdown table. JSON to "flat" view

Опис:

Актор - User (Користувач) - взаємодіє з системою та має доступ до таких функцій:

- Зареєструватися
- Увійти
- Завантажити JSON файл
- Створити новий JSON
- Валідувати JSON
- Форматувати JSON
- Мініфікувати JSON
- Конвертувати JSON в XML/CSV
- Зберегти результат
- Переглянути історію операцій
- Вийти з системи

Актор - Admin (Адміністратор) - має всі функції користувача, а також додаткові можливості адміністрування:

- Увійти як адміністратор
- Керувати користувачами
- Переглядати системні логи
- Налаштовувати параметри системи
- Керувати шаблонами JSON

Варіанти використання:

1. Управління обліковим записом

- **Реєстрація користувача** – створення нового облікового запису в системі.
- **Авторизація користувача** – вхід до системи за допомогою логіна та пароля.
- **Вихід із системи** – завершення сеансу користувача.
- **Управління ролями** – адміністратор може призначати або змінювати ролі користувачів.

2. Робота з JSON документами

- **Створення документа** – користувач створює новий JSON документ.
- **Редагування документа** – внесення змін у існуючий JSON документ.
- **Видалення документа** – повне видалення JSON документа з системи.
- **Перегляд документів** – відображення списку всіх документів користувача із можливістю сортування.
- **Валідація JSON** – перевірка правильності структури документа.
- **Форматування/мініфікація JSON** – зміна структури документа для читабельності або стискання.

3. Організація документів

- **Створення шаблонів** – користувач може створювати шаблони JSON для повторного використання.
- **Прив’язка шаблонів до документів** – швидке застосування структури шаблону до нового документа.
- **Сортування за датою чи шаблоном** – швидка навігація між документами.

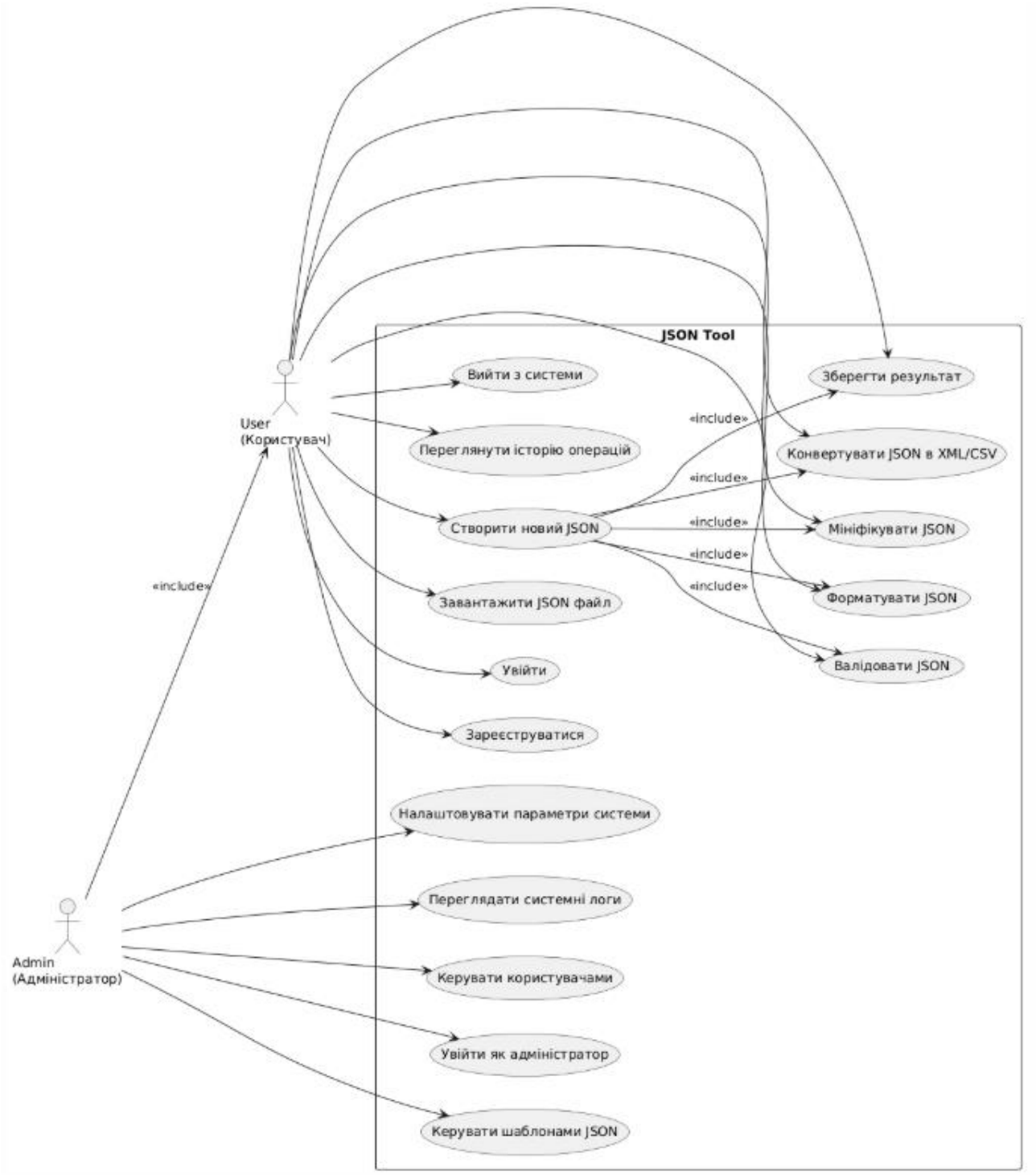
4. Пошук і фільтрація

- **Пошук за назвою документа** – знаходження документів за ключовими словами у назві або вмісті.
- **Фільтрація за користувачем або шаблоном** – перегляд лише потрібних документів.

5. Історія та моніторинг

- **Запис операцій** – система зберігає історію всіх дій користувача над JSON документами.
- **Перегляд історії** – користувач або адміністратор можуть перевіряти виконані операції.
- **Аналіз результатів операцій** – перевірка успішності обробки документів.

Діаграма варіантів використання:



Клас User моделює користувача системи JSON Tool. Він відповідає за базові дії з системою та JSON документами.
Атрибути:

- id: int — унікальний ідентифікатор користувача
- username: String — логін користувача
- password: String — пароль користувача
- createdAt: Date — дата створення акаунта

Операції:

- login(): boolean — авторизація в системі
- logout(): void — вихід із системи
- register(): boolean — реєстрація нового користувача

Клас Role моделює роль користувача в системі (наприклад, "Admin", "User").
Атрибути:

- id: int — унікальний ідентифікатор ролі
- name: String — назва ролі

Клас JSONDocument моделює JSON документ у системі.
Атрибути:

- id: int — унікальний ідентифікатор документа
- name: String — назва документа
- content: String — вміст JSON
- createdAt: Date — дата створення
- modifiedAt: Date — дата останньої модифікації
- userId: int — власник документа

Операції:

- save(): boolean — збереження документа
- load(): JSONDocument — завантаження документа
- delete(): boolean — видалення документа
- validate(): boolean — перевірка коректності JSON

Клас OperationHistory зберігає історію дій користувачів над JSON документами.
Атрибути:

- id: int — унікальний ідентифікатор запису
- userId: int — користувач, що виконав операцію
- documentId: int — документ, над яким була операція
- operationType: String — тип операції (створення, редагування, видалення)
- executedAt: Date — дата та час виконання
- result: String — результат виконання

Клас Template моделює шаблони JSON, які можна повторно використовувати.
Атрибути:

- id: int — унікальний ідентифікатор шаблону
- name: String — назва шаблону
- schemaContent: String — структура або вміст шаблону
- createdBy: int — користувач, що створив шаблон
- createdAt: Date — дата створення

Клас JSONProcessor реалізує основну логіку обробки JSON документів.
Атрибути:

- currentJSON: String — поточний JSON документ

Операції:

- validate(json: String): boolean — перевірка коректності JSON
- format(json: String): String — форматування JSON
- minify(json: String): String — мініфікація JSON
- parse(json: String): Object — перетворення JSON у об’єкт

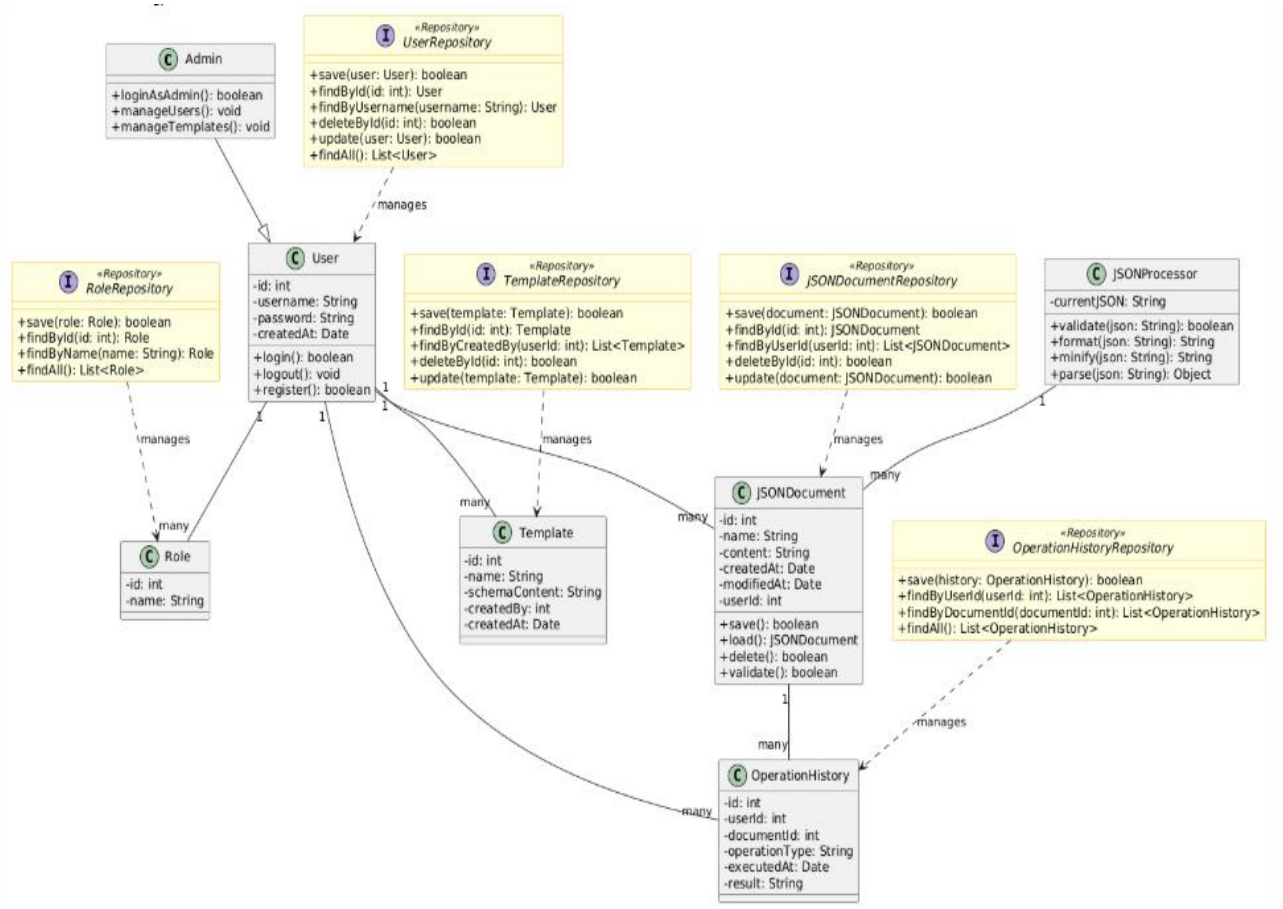
Клас Admin моделює адміністратора системи. Наслідує User та має додаткові можливості.
Операції:

- loginAsAdmin(): boolean — вхід з правами адміністратора
- manageUsers(): void — керування користувачами
- manageTemplates(): void — керування шаблонами

Взаємозв’язки:

- User може мати багато JSONDocument та багато OperationHistory
- User може мати багато Role (багато-до-багатьох)
- User може створювати багато Template
- JSONDocument пов’язаний з багатьма OperationHistory
- Admin наслідує User
- JSONProcessor обробляє JSONDocument

Діаграма класів предметної області



3 варіанти використання та сценарії використання:

Сценарій 1: Валідувати JSON

Передумови:

- Користувач увійшов до системи
- JSON контент готовий для валідації (завантажений або введений)

Постумови:

- Система показує результат валідації (валідний/невалідний)
- Якщо JSON невалідний - показуються помилки

Взаємодіючі сторони:

- Користувач, клас JSONProcessor

Короткий опис:

- Користувач перевіряє чи є JSON документ синтаксично правильним

Основний перебіг подій:

- Користувач відкриває функцію валідації

- Вводить або завантажує JSON контент
- Натискає кнопку "Валідувати"
- Система викликає JSONProcessor.validate()
- Система відображає результат валідації

Винятки:

- Якщо JSON порожній - виводиться повідомлення про помилку
- Якщо сталася системна помилка - показується загальне повідомлення про помилку

Примітки:

- Валідація відбувається в реальному часі при введенні

Сценарій 2: Створення нового JSON документа

Передумови:

- Користувач увійшов до системи
- Користувач має доступ до інтерфейсу створення документа

Постумови:

- JSON документ збережений у системі
- Користувач може переглянути, редагувати або видалити документ

Взаємодіючі сторони:

- Користувач, клас JSONDocument, клас JSONProcessor

Короткий опис:

- Користувач створює новий JSON документ та зберігає його у системі

Основний перебіг подій:

1. Користувач відкриває форму створення документа
2. Вводить назву документа та JSON контент
3. Натискає кнопку "Зберегти"
4. Система викликає JSONDocument.save()
5. Документ додається до списку користувача
6. Система підтверджує успішне збереження

Винятки:

- Якщо JSON некоректний – система повідомляє про помилку
- Якщо документ з такою назвою вже існує – пропонується змінити назву

Примітки:

- Можливе використання шаблону для швидкого створення документа

Сценарій 3: Перегляд історії операцій над документом

Передумови:

- Користувач увійшов до системи
- Існують виконані операції над документами користувача

Постумови:

- Користувач бачить список усіх операцій над документами
- Можливість фільтрувати або сортувати записи

Взаємодіючі сторони:

- Користувач, клас OperationHistory, клас JSONDocument

Короткий опис:

- Користувач переглядає історію змін та операцій над своїми JSON документами

Основний перебіг подій:

1. Користувач відкриває розділ "Історія операцій"
2. Система витягує записи з OperationHistory
3. Система відображає список операцій з інформацією: документ, тип операції, дата та результат
4. Користувач може застосувати фільтри за датою, документом або типом операції

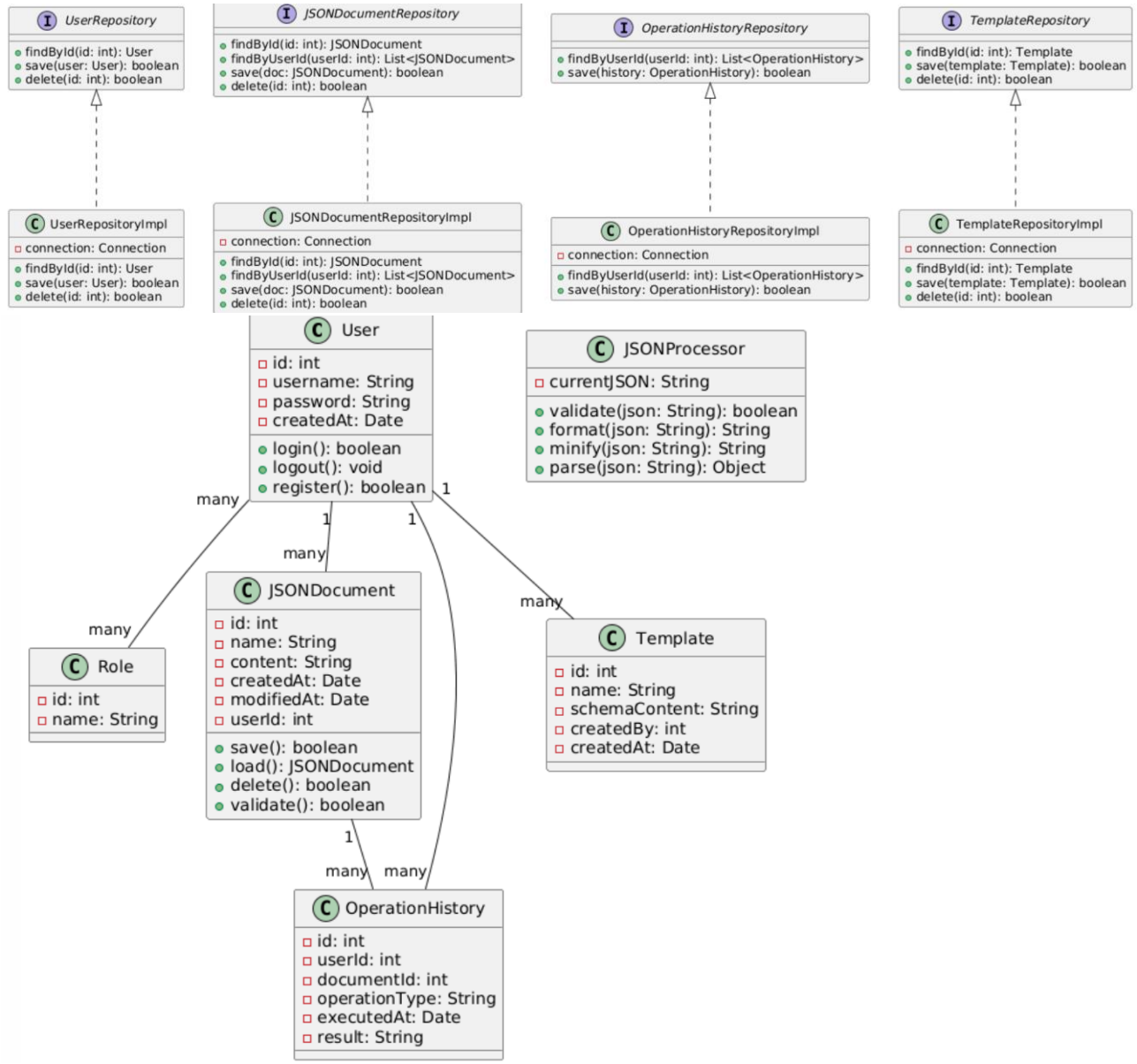
Винятки:

- Якщо історії немає – відображається повідомлення "Записів не знайдено"
- Якщо сталася системна помилка – показується загальне повідомлення

Примітки:

- Адміністратор може переглядати історію всіх користувачів

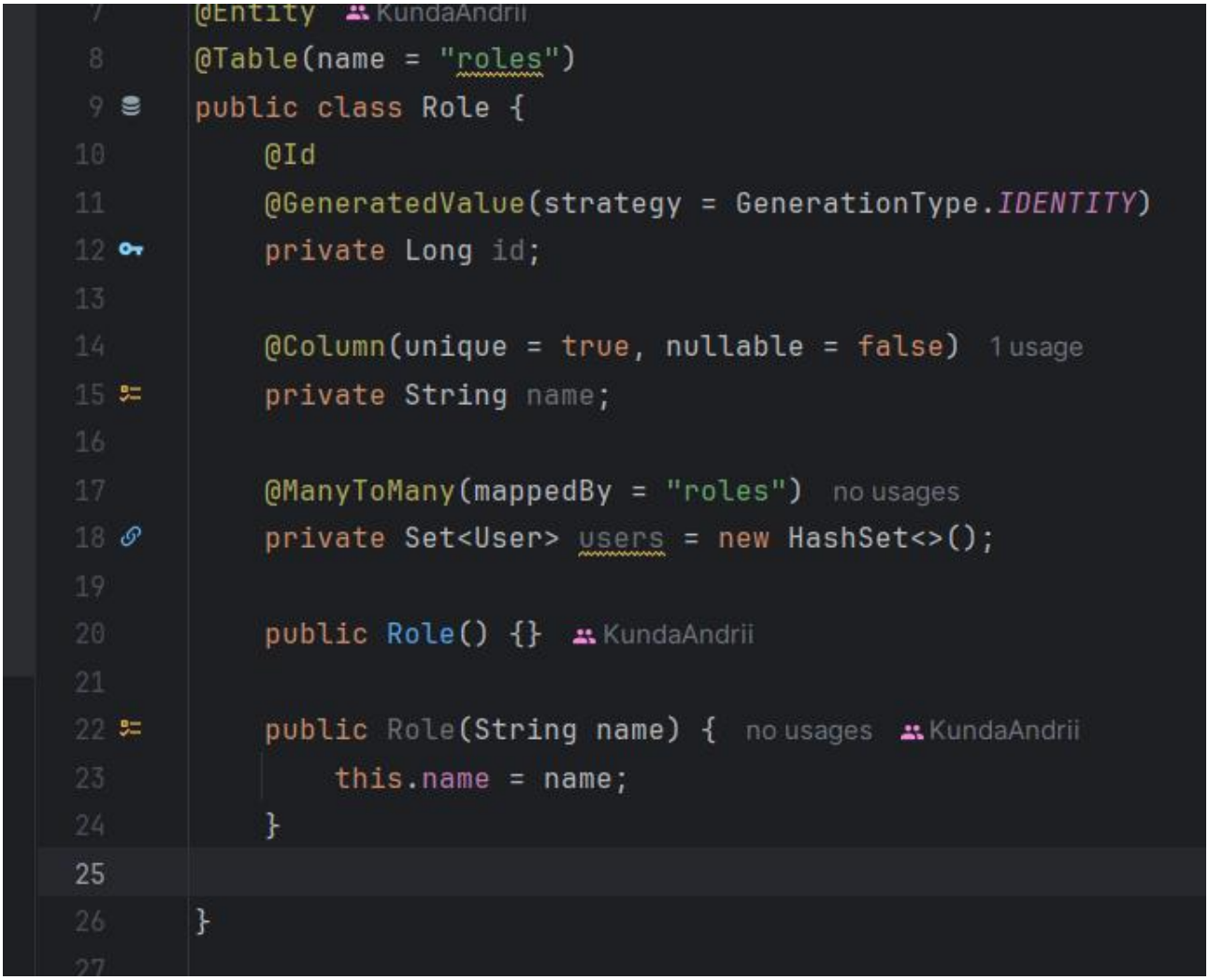
На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних:



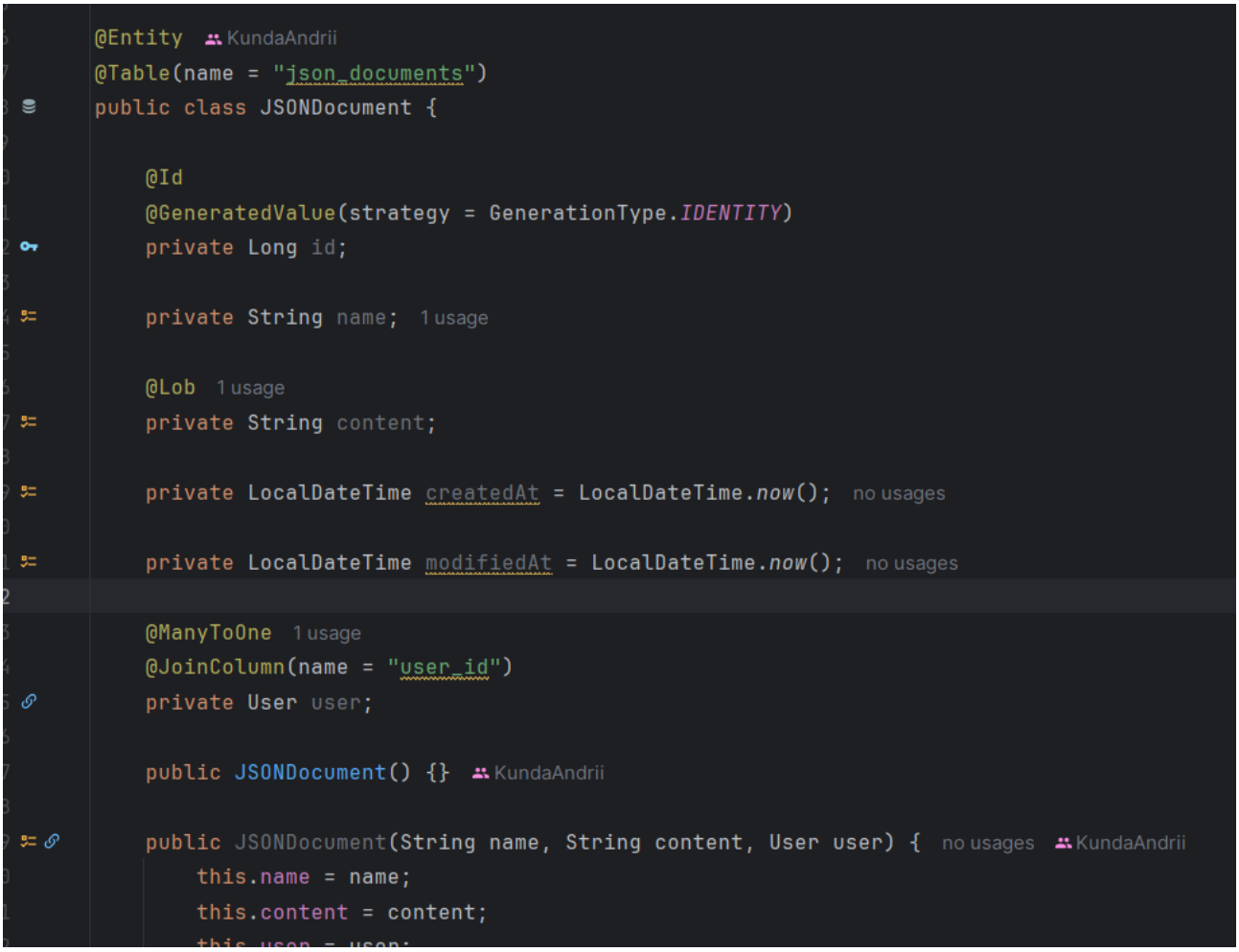
Код реалізації:

```java
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 50)  1 usage
    private String username;

    @Column(nullable = false)  1 usage
    private String password;

    private LocalDateTime createdAt = LocalDateTime.now();  no usages

    // Зв'язок з ролями (many-to-many)
    @ManyToMany(fetch = FetchType.LAZY)  no usages
    @JoinTable(
            name = "user_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set<Role> roles = new HashSet<>();

    // Зв'язок з документами (one-to-many)
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)  no usages
    private Set<JSONDocument> documents = new HashSet<>();

    public User() {}  KundaAndrii

    public User(String username, String password) {  no usages  KundaAndrii
        this.username = username;
        this.password = password;
    }

}
```

User.java


@Entity KundaAndrii
@Table(name = "roles")
public class Role {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
@Column(unique = true, nullable = false) 1 usage
private String name;
@ManyToMany(mappedBy = "roles") no usages
private Set<User> users = new HashSet<>();
public Role() {} KundaAndrii
public Role(String name) { no usages KundaAndrii
this.name = name;
}
}

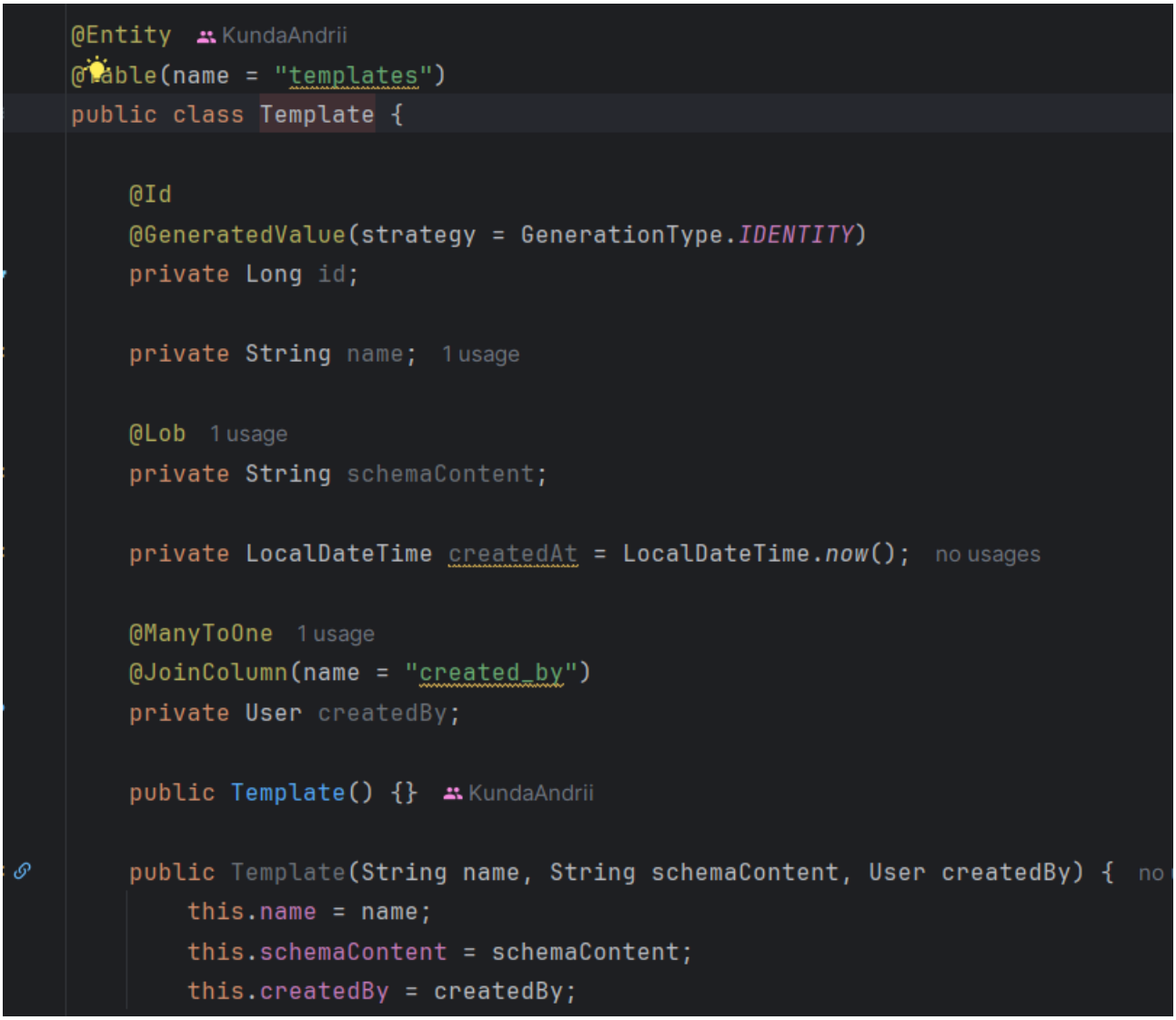
Role.java



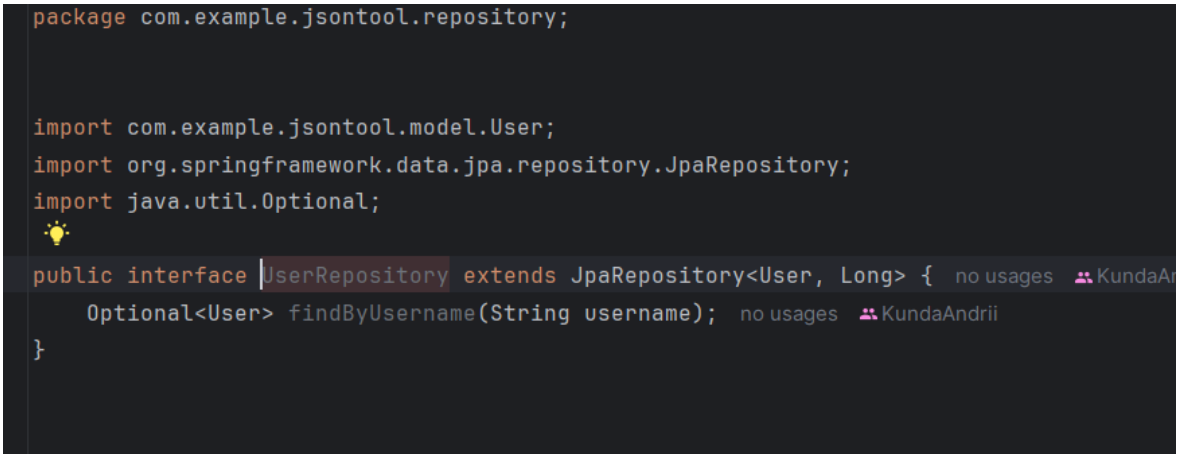
JSONDocument.java

```java
@Table(name = "operation_history")
public class OperationHistory {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String operationType;  1 usage

    private LocalDateTime executedAt = LocalDateTime.now();  no usages

    @Lob  1 usage
    private String result;

    // Зв'язки
    @ManyToOne  1 usage
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne  1 usage
    @JoinColumn(name = "document_id")
    private JSONDocument document;

    public OperationHistory() {}  KundaAndrii

    public OperationHistory(String operationType, String result, User
        this.operationType = operationType;
        this.result = result;
        this.user = user;
        this.document = document;
    }
```

OperationHistory.java



Template.java



UserRepository.java

```java
import com.example.jsontool.model.Role;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RoleRepository extends JpaRepository<Role, Long> {  no usages
    Role findByName(String name);  no usages  KundaAndrii
}
```

RoleRepository.java

```java
package com.example.jsontool.repository;


import com.example.jsontool.model.JSONDocument;
import com.example.jsontool.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface JSONDocumentRepository extends JpaRepository<JSONDocument, Long> {  no usages
    List<JSONDocument> findByUser(User user);  no usages  KundaAndrii
}
```

JSONDocumentRepository.java

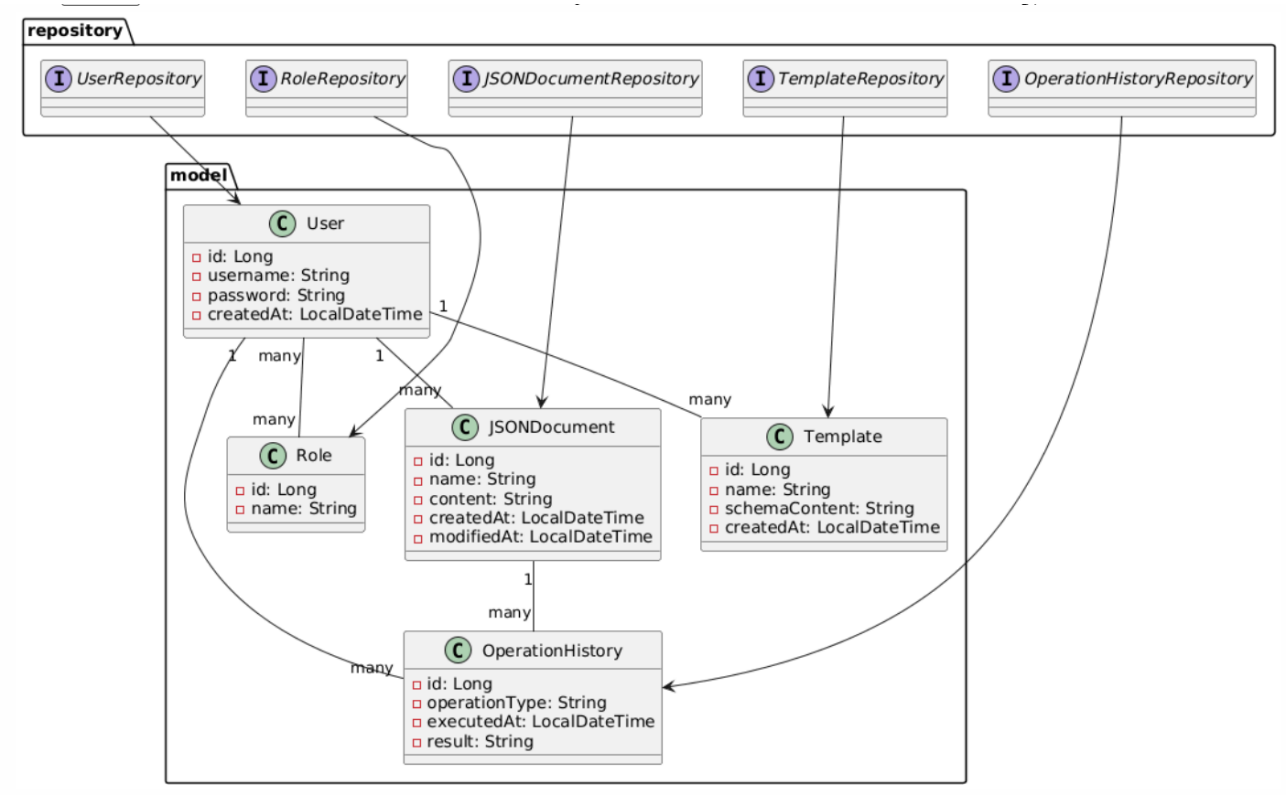
```java
import com.example.jsontool.model.OperationHistory;
import com.example.jsontool.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface OperationHistoryRepository extends JpaRepository<OperationHistory, Long> {
    List<OperationHistory> findByUser(User user);  no usages  KundaAndrii
}
```

OperationHistoryRepository.java

```java
import com.example.jsontool.model.Template;
import com.example.jsontool.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface TemplateRepository extends JpaRepository<Template, Long> {
    List<Template> findByCreatedBy(User user);  no usages  KundaAndrii
}
```

TemplateRepository.java

6. Нарисувати діаграму класів для реалізованої частини системи.



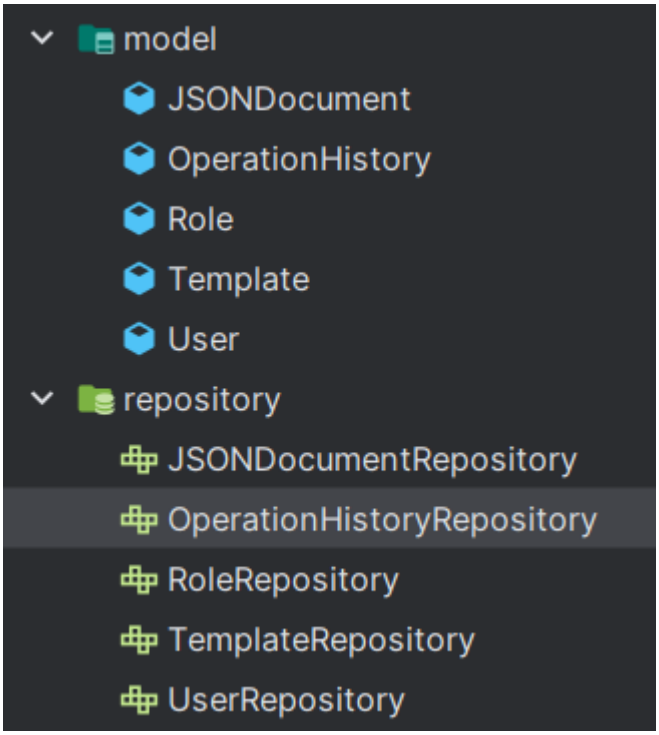


Рис. 9 – Реалізовані класи

Висновок:

У межах лабораторної роботи я спроєктував систему JSON Tool, яка дозволяє користувачам працювати з JSON документами. Були створені діаграми варіантів використання та класів, розроблені сценарії використання для трьох основних функцій системи: валідації JSON, конвертації. Також була спроєктована структура бази даних та реалізовані основні класи з використанням шаблону Repository для ефективної взаємодії з базою даних. Система передбачає як базовий функціонал для користувачів, так і розширені можливості для адміністраторів.

Відповіді на контрольні питання

1. Що таке UML?

UML (Unified Modeling Language) — це уніфікована мова моделювання, яка використовується для графічного представлення, документування та

проектування програмних систем. Вона допомагає описати структуру, поведінку
та взаємодію компонентів системи.

2. Що таке діаграма класів UML?

Діаграма класів UML — це статична діаграма, яка показує класи системи, їх атрибути, методи та зв’язки між класами. Вона використовується для моделювання структури об’єктно-орієнтованих програм.

3. Які діаграми UML називають канонічними?

Канонічні (основні) діаграми UML — це ті, що найчастіше використовуються для опису системи:

• Діаграма класів
• Діаграма варіантів використання (Use Case)
• Діаграма послідовності (Sequence)
• Діаграма станів (State)
• Діаграма активностей (Activity)

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання (Use Case Diagram) показує взаємодію користувачів (акторів) з системою через варіанти використання. Вона відображає, хто і як використовує систему, без деталізації внутрішньої реалізації.

5. Що таке варіант використання?

Варіант використання (Use Case) — це послідовність дій, які система виконує для досягнення певної мети користувача. Наприклад, “Реєстрація користувача” або “Надіслати повідомлення”.

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання можуть бути:

• Association (асоціація) — зв’язок між актором і варіантом використання.

• Include (включення) — один варіант використання завжди виконує інший.
• Extend (розширення) — додатковий варіант, який може виконуватися у певних
умовах.
• Generalization (успадкування) — актор або варіант використання наслідує інший.
7. Що таке сценарій?
Сценарій — це конкретна реалізація варіанту використання, тобто послідовність
кроків, які відбуваються під час виконання дії користувачем та системою.
8. Що таке діаграма класів?
Діаграма класів — це графічне представлення класів, їх атрибутів і методів, а також зв’язків між класами. Вона описує структуру системи.
9. Які зв’язки між класами ви знаєте?
Основні типи зв’язків:
• Асоціація (Association) — загальний зв’язок між класами.
• Агрегація (Aggregation) — «має»; клас складається з інших, але частини можуть існувати окремо.
• Композиція (Composition) — сильніша агрегація; частини не можуть існувати без цілого.
• Успадкування (Generalization) — один клас наслідує властивості іншого.
• Залежність (Dependency) — один клас використовує інший тимчасово.
10. Чим відрізняється композиція від агрегації?
• Агрегація: частини можуть існувати без цілого.
• Композиція: частини не можуть існувати без цілого; знищення цілого знищує частини.
11. Чим відрізняється агрегація від композиції на діаграмах класів?
• Агрегація позначається порожнім ромбом на стороні цілого.

• Композиція позначається заповненим ромбом на стороні цілого.
12. Що являють собою нормальні форми баз даних?
Нормальні форми — це правила організації таблиць БД для уникнення надлишковості та аномалій при оновленні даних. Основні:
• 1НФ — всі атрибути атомарні.
• 2НФ — всі неключові атрибути залежать від усього первинного ключа.
• 3НФ — немає транзитивних залежностей між неключовими атрибутами.
13. Що таке фізична і логічна модель БД?
• Логічна модель — структура даних з таблицями, полями та зв’язками, незалежно від СУБД.
• Фізична модель — конкретна реалізація БД у СУБД, включає типи даних, індекси, обмеження, фізичне зберігання.
14. Який взаємозв'язок між таблицями БД та програмними класами?

Кожна таблиця БД часто відповідає класу у програмі, а рядки таблиці — об’єктам класу. Поля таблиці відображаються як атрибути класу, а зв’язки між таблицями як зв’язки між класами.